Byzantine-Resilient Routing and Key Management Protocols using Network Coding

Cristina Nita-Rotaru

Department of Computer Science and CERIAS
Purdue University
Acknowledgment

This work was funded by NSF, Secure Networking Using Network Coding

Relevant publications

- **Node-Capture Resilient Key Establishment in Sensor Networks: Design Space and Protocols.** Andrew Newell, Hongyi Yao, Alex Ryker, Tracey Ho, and Cristina Nita-Rotaru. ACM Computing Surveys, Jan. 2015


Overarching goal:

Create and build distributed systems and network protocols that achieve **security**, **availability**, and **performance** in spite of **misconfigurations, failures, and attacks**

Approach:

Combine theoretical principles and experimental methodologies from distributed systems, cryptography, networking, information theory, and machine learning
The Internet of everything is here ...

- Computing services
  - Everything is connected
  - Many types of devices
  - Tremendous amount of data
  - Available via cloud computing, accessed via personal devices

- Higher expectations
  - Services must be available 24h, working correctly 100% of the time
  - Data-centric business, policy decisions

Users called 911 because Facebook was down !!!
What does it mean for security

- Large number of devices with different capabilities and vulnerabilities managed by different entities
  - Higher chances that some system components are going to be compromised
  - *The next attack is going to come from your kitchen*
- Subset of computing systems or protocol participants controlled by an adversary can influence
  - Communication and availability
  - Data quality, processing, and learning

Designing systems resilient to only outsider attackers no longer sufficient, need for insider-resilient systems
Seeing the world through a Byzantine lens

- An insider can not be trusted to correctly generate or process data (i.e. lie):
  - **Trusting info limitations**
    - Many insider nodes collude
    - Not enough history is available
    - Single source of information

- An insider can not be trusted to correctly deliver data:
  - **Disseminating info limitations**
    - Lack of non-adversarial paths
    - Not enough redundancy
    - Correlated failures
Network coding: A New paradigm

- **Key principle:** packet mixing at intermediate nodes

- **Benefits:** Higher throughput, reliability, robustness, energy efficiency

- **Applications:** wireless unicast and multicast, p2p storage and content distribution, delay-tolerant networks, vehicular networks
Network coding in wireless networks

- Opportunities
  - Broadcast advantage
  - Opportunistic listening

- Benefits
  - Improved throughput
  - Reduced delay
  - Improved reliability
This talk

- Network coding under attack:
  - Pollution attacks in intra-flow network coding
- Network coding to the rescue:
  - All pairwise and connected graph key management resilient to node capture
Wireless network coding systems

- Intra-Flow Network Coding
  - Mix packets within individual flows
  - Examples: [Park; 2006], MORE [Chachulski; 2007], [Zhang; 2008a], [Zhang; 2008b], MIXIT [Katti; 2008], [Lin; 2008]

- Inter-Flow Network Coding
  - Mix packets across multiple flows
  - Examples: COPE [Katti; 2006], DCAR [Le; 08], [Das; 2008], [Omiwade; 2008a], [Omiwade; 2008b]
Intra-flow network coding

1. Divide plain packets into generations
2. Broadcast coded packets

Source node
Plain packets
... p_1, p_2, ... p_n

Forwarder nodes

1. Buffer overheard coded packets
2. Broadcast new coded packets

Receiver node
1. Buffer coded packets
2. Decode packets
3. Send ACK to source

Cristina Nita-Rotaru
Packet coding and decoding

- \( \mathbf{p}_i = (p_{i1}, p_{i2}, \ldots, p_{im})^T, p_{ij} \in \mathbb{F}_q \)
- \( G = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n] \)
- Coding with random linear combination
  - \( \mathbf{c} = (c_1, c_2, \ldots, c_n), c_i \in \mathbb{F}_q \)
  - \( \mathbf{e} = c_1 \mathbf{p}_1 + c_2 \mathbf{p}_2 + \ldots + c_n \mathbf{p}_n = G\mathbf{c} \)
- Decoding
  - Given \( n \) linearly independent coded packets \( (\mathbf{c}_1, \mathbf{e}_1) \ldots (\mathbf{c}_n, \mathbf{e}_n) \)
    solve a system of linear equations
- Attacks
  - **Packet Pollution**: injecting incorrect packets
Pollution attacks

Pollution attacks are attacks where attackers inject polluted coded packets into the network.

A coded packet \((c, e)\) is a polluted coded packet if

\[
c = (c_1, c_2, \ldots, c_n), \quad c_i \in \mathbb{F}_q
\]

but

\[
e \neq c_1p_1 + c_2p_2 + \ldots + c_n p_n
\]

Generic attack to any network coding system
Impact of pollution attacks

Source node

Generation
$p_1, p_2, \ldots, p_n$

Forwarder nodes

Receiver node

Epidemic attack propagation
Prior work

- Cryptographic approaches [Krohn; 2004], [Li; 2006], [Charles; 2006], [Zhao; 2007], [Yu; 2008], [Boneh; 2009]
  - Homomorphic digital signatures or hash functions
  - *Too expensive computationally*

- Information theoretic approaches [Ho; 2004], [Jaggi; 2007], [Wang; 2007]
  - Coding redundant information
  - *Low achievable throughput*

- Network error correction coding [Yeung; 2006], [Cai; 2006], [Silva; 2007], [Koetter; 2008]
  - Using error correction coding techniques
  - *Limited error correction capability, unsuitable for adversarial environment*
Throughput CDF when no attack happens

The high overhead of crypto-based schemes render them impractical for wireless networks.
## Our approach

<table>
<thead>
<tr>
<th>Non-cryptographic checksum created by the source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Based on lightweight random linear transformations</td>
</tr>
<tr>
<td>Carries the timestamp of when it was created</td>
</tr>
<tr>
<td>Disseminated by the source in an authenticated manner</td>
</tr>
<tr>
<td>Not pre-image or collision resistant!</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>security relies on time asymmetry checksum verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>A node verifies a packet against a checksum that is created after the packet is received</td>
</tr>
</tbody>
</table>
Our approach: Example

Attacker can not inject a checksum or modify timestamp because checksum is signed by source.

Packet p will be verified against CS₂ and not CS₁. The attacker does not gain anything by observing CS₁.
DART and EDART

- **DART**
  - Forwarder nodes buffer packets checksum verification
  - Only verified packets are combined to form new packets for forwarding
  - Polluted packets are dropped at first hop, eliminating epidemic propagation

- **EDART**
  - Improves performance with optimistic forwarding
Checksum computation and verification

- A generation of packets $G = [p_1, p_2, \ldots, p_n]$

**Checksum computation**

- Compute $H_s$, a random $b \times m$ matrix from a seed $s$
- Compute the checksum

\[ \text{CHK}_s(G) = H_s G \]

- $b$ is a system parameter that trades off security and overhead

**Checksum verification**

Given $\text{CHK}_s(G)$, $s$ and $t$, check if a coded packet $(c, e)$ is valid

- Check

\[ \text{CHK}_s(G) \ c = H_s \ e \]

- Why?

\[ \text{CHK}_s(G)e = (H_s G)e = H_s(Gc) = H_s \ e \]

- No false positive, may have false negative
Batch Checksum Verification

- Verify a set of coded packets \{((c_1, e_1), \ldots, (c_k, e_k))\} at once

For higher accuracy, we can repeat the procedure
**DART Algorithm**

**Source node**
- Disseminate coded packets as usual
- Periodically disseminate a signed random checksum (CHK, s, t)

**Forwarder node**
- **On sending a packet**
  Code packets in verified set
- **On receiving coded packet p**
  Add p to unverified set, record receive time
- **On receiving checksum (CHK, s, t)**
  Verify packets in unverified set with receive time before t

---

### Diagram:
- **Source node** S
- **Unverified** and **Verified** sets
- **Check sum**
- **Forwarder node** D
- **Receiver node** R

---

Cristina Nita-Rotaru
DART Overhead Analysis

- Computation overhead
  - Checksum computation
    - $\text{CHK}_s(G) = H_s G$
  - Checksum verification
    - $\text{CHK}_s(G)c = H_se$

- Communication overhead
  - Dissemination of checksum packet $(\text{CHK}_s(G), s, t)$
    - $s$: random seed, e.g. 4 bytes
    - $t$: timestamp, e.g. 4 bytes
    - $\text{CHK}_s(G)$: $b \times n$ matrix over $F_q$
      - Example: $b=2$, $n=32$, $q=2^8$, $\text{CHK}_s(G)$ is 64 bytes
DART security analysis

Claim

- The probability that a polluted packet can pass the checksum verification is \(1/q^b\)
- In batch verification, the probability that a polluted packet passes \(w\) independent batch verification is \(1/q^b + 1/q^w\)

Example: \(q = 2^8, b = 2\)
- 1 in 65536 polluted packets can pass first hop neighbor
- 1 in over 4 billion polluted packets can pass second hop neighbor
EDART

- DART delays packets for verification, increasing latency

Ideally,
- Delay polluted packets for verifying
- Forward correct packets without delay

But,
- We do not know which packets are correct and which are polluted
EDART overview

- Packets are always verified BUT
- Nodes "closer" to the attacker delay packets for verification
- Nodes "farther" away from the attacker forward packets without delay and will verify them when possible

- Polluted packets are restricted to a region around the attacker
- Correct packets are forwarded without delay
- In case of no attack, all packets are forwarded without delay – almost no impact on performance
How to decide when to delay?

- $h_{uv}$: Add a hop count that captures the number of hops a packet has traveled since the last verification.
  - All verified packets will have $h_{uv}$ set to 0.
  - **Packets that traveled less than $\delta$ hops will be forwarded without delay, otherwise a node delays them.**
  - When coding a new packet, set $h_{uv} = h_{\text{max}} + 1$ to be the maximum $h_{uv}$ in the packets used to create the new packet.
  - If pollution was detected, the node will switch for a time proportional with how big $h$ is to delaying all packets.
EDART Algorithm

**Forwarder Node State**
- **Node mode**
  - Delay mode
  - Forward mode
- **Delay forward timer**
  - $C_v = 0 \rightarrow$ in forward mode

**Packet Field**
- $h_{uv}$ the number of hops a packet had traveled since its last verification checksum

**Forwarder Node Algorithm**
- **On sending a packet**
  Code packets in forward set
- **On receiving coded packet $p$**
  - if $C_v > 0$ or $h_{uv} \geq 6$
    - Add $p$ to delay set
  - else
    - Add to forward set
- **On receiving checksum (CHK, s, t)**
  - Verify unverified packets (delayed or not)
  - if detecting a polluted packet $p$
    - Increase $C_v$ by $\alpha \left(1 - h_{uv}/\delta\right)$
  - else if $C_v > 0$
    - Decrease $C_v$ by 1

---

Cristina Nita-Rotaru
EDART security analysis

- Maximum pollution scope
  - Bounded by $\delta + 1$
- Average pollution scope
  - Bounded by $\delta/\alpha$
- Maximum pollution success frequency
  - Bounded by $\delta/\alpha$
- Unnecessary delay
  - Nodes at $i$ hops away from the attacker
    $(2 < i < \delta-h-1): \alpha(1 - (h+i)/\delta)$
  - Nodes more than $\delta-h-1$ hops away: 0

The selection of $\delta$ and $\alpha$ trades off security and performance
Experimental evaluations

- Network coding system: MORE
- Simulator: Glomosim
- Trace driven physical layer
  - MIT Roofnet trace
- MORE setup
  - $GF(2^8)$, generation size 32, packet size 1500 bytes
- Defense setup
  - RSA-1024 digital signature
  - Checksum size parameter $b = 2$
  - EDART setup $\delta = 8$, $\alpha = 20$
Impact of pollution attacks

Even a single pollution attacker can be extremely detrimental!

Pollution intensity (PI): number of polluted packets injected per packet received

Throughput CDF under a single pollution attacker with various pollution intensity

Cumulative Fraction of Flows

0 0.2 0.4 0.6 0.8 1

Throughput (kbps)

0 500 1000 1500 2000 2500

No Attack
PI: 1.00
PI: 0.20
PI: 0.10
PI: 0.05

97%
Effectiveness of DART and EDART

**Ideal Defense:** defense scheme that drops polluted packets with zero overhead

Both DART and EDART are very effective against pollution attacks
Performance in benign networks

Throughput CDF

Latency CDF

Both DART and EDART have good performance. EDART has almost zero performance impact.
Overhead of DART and EDART

Both DART and EDART incur small bandwidth and computation overhead

Only 2% of system throughput
Null Keys

- Valid coded packets belong to a subspace $A$
- A null key $K$ is a subspace of $N(A)$, $N(A)$ is the null space of $A$
  - If $c$ in $A$ then $c \times K = 0$
  - If $c$ not in $A$ then $c \times K \neq 0$ with high probability
- Low computational overhead for verification compared to digital signature/hash schemes
A basic approach

- Source distributes null keys to some forwarders
- Forwarders exploit subspace property of null keys to combine their null keys for other forwarders
- Path diversity ensures a forwarder's null keys do not span the space of a downstream node's null keys

- However
  - No path diversity in wireless
  - Null keys are very large
Our Approach

Splitting the null keys

- **Generation independent part**
  - Large (7340 bytes in our typical scenario)
  - Constant for multiple generations
- **Generation dependent part**
  - Small (160 bytes in our typical scenario)
  - Updated each generation
- **Source distributes large independent parts once**
- **Source periodically updates smaller dependent parts**

Advantages

- Low communication overhead
- No need for forwarders, source can send the key updates
### Splitting Null Keys

#### Notation

- \( n \) – number symbols in coding header
- \( m \) – number symbols of coded data
- \( w \) – Size of null key
- \( K \) – null key \(((n+m) \times w)\) matrix\)
- \( K_d \) – generation dependent null key \((n \times w)\) matrix\)
- \( K_i \) – generation independent null key \((m \times w)\) matrix\)
- \( X \) – data for generation \((n \times m)\) matrix\)

#### Key Splitting

1) Initialize \( K_i \) randomly
2) \( K_d := X \times K_i \)
3) \( K = \left[ K_d^T \mid K_i^T \right]^T \)

#### Packet Verification

- \( c \times K = 0 \) if \( c \) from \( X \)
- \( n << m \) so \( K_d << K_i \)
Comparison with pollution defenses

- **SNK** – Split Null Keys
- **DART** – Wireless defense based on time-sensitive checksums
- **KFM** – Representative crypto-based scheme
- **MORE** – Network coding without defense overhead
- **HOMOMAC-x** – MAC-based scheme resilient to $x$ attackers

- SNK outperforms other defenses
  - Low computational overhead
  - No delaying of packets
  - Not sensitive to multiple attackers
Retains coding gains

- **SNK** – Split Null Keys
- **MORE** – Network coding without defense overhead
- **ARAN** – Secure best-path-routing protocol
- SNK retains coding gains of MORE while providing defense against attackers
This talk

- Network coding under attack:
  - Pollution attacks in intra-flow network coding
- Network coding to the rescue:
  - All pairwise and connected graph key management resilient to node capture
Key distribution in wireless network

How to bootstrap trust in a wireless (sensor) network?

- Establish secret keys
  - All pairwise keys: Symmetric keys are established between every pair of nodes in the network
  - Connected graph: Enough keys are established to ensure that the network graph is connected

- By using different types of communication
  - Direct: nodes communicate directly
  - Multi-hop: nodes communicate through intermediate nodes
    - Single path
    - Multi-path
Resilience to node capture

How many keys get compromised when a node is captured?

- All nodes share the same key
  - Compromise of a node means compromise of the entire network
- Pairwise keys
  - Only the keys shared by the compromised node with other nodes in the network get compromised
- Connected graph
  - Each node requires fewer keys, but can result in high communication overhead as the shortest path over secure links may be larger than the shortest path over all possible links.
Typical key establishment steps

- Network operator first initializes each sensor with a set of secret keys chosen from a large pool.
- Sensor nodes are dispersed randomly and uniformly in an environment.
- Sensor nodes discover their physical neighbors determined by a fixed transmission range.
- Pairs of physical neighbors aim to establish a secret key by using their pre-shared keys:
  - communicating directly
  - communicating with other nodes over multi-hop paths.
Factors in the design space

- Secrecy and correctness (i.e. integrity, i.e. resilience) of the keys – depending on adversarial model during the key establishment

- Memory constraints
  - How many keys does a node store?

- Network resilience to attacks
  - How many secure links (secret keys) are compromised when a node is compromised: security constraints

- Communication overhead
  - Communication overhead needed to establish keys and communicate securely
Our approach

- New coding technique
  - Single-path scheme
  - Multi-path scheme for both connected component and all pairwise keys
  - Provides both secrecy and correctness
  - Maximal rate


- Assume attackers are present during key establishment
Coding technique

- Secrecy and correctness under bounded number of adversaries

```latex
\begin{align*}
    H_i &= [h_i^j : j = 1, 2, \ldots, n, j \neq i] \\
    P_i &= [H_i, D_i, m_i]
\end{align*}
```

![Diagram](https://via.placeholder.com/150)
Evaluation goals

- How do changes in the proportion of compromised nodes, available memory and network size affect the resilience to node compromises for each scheme?
- How do changes in the network size and density affect the communication overhead for each scheme?
- How do all pairwise keys schemes compare with connected graph schemes?
- How do changes in the number of disjoint paths for the multi-path schemes affect overhead and security?
All pairwise: Proportion of insecure links

Figure 6 shows protocols under varying degrees of attack by altering the proportion of compromised nodes. These protocols are all using the same memory budget at each node (memory equivalent to 60 symmetric secret keys), so we are observing the difference in how each protocol resists colluding attackers. The D-AP is an all-or-none type of protocol where either all links are secure or no links are secure. The main drawback is that the threshold of number of attackers where all links are insecure is quite low, and it is analytically 0.06 proportion of compromised nodes. As a result, at very low proportion of compromised nodes, the D-AP actually performs the best, but it is quickly overtaken by the path based protocols. The difference between the P-AP and MP-AP protocols is roughly 0.2 more proportion of links being insecure. This observation shows that the multiple paths are indeed increasing resilience as more compromised nodes must be compromised before all multiple paths are compromised. We investigate more later in Section 7.4 the difference in the number of paths (i.e., MP-AP-3 vs MP-AP-5) which favors more paths when more memory exists in the network.

Figure 7 shows protocols under varying available memory to initialize nodes with key secret information. We see the all or nothing security of D-AP, which requires at least memory equal to 100 secret keys to ensure all links are established securely which is due to the 10% compromised nodes in this scenario. The path and multi-path based protocols are capable of establishing many links securely despite very low amounts of memory. They have increased resilience as memory improves. The P-AP protocol is initialized with a grid based logical topology which has 3-dimensions when memory is 20 keys, but is 2-dimensions for the memory value of 35 keys and larger, so resilience does not increase well when more memory is added beyond 35 keys. In the MP-AP case, the random based logical topology allows additional memory to improve resilience as more, shortest disjoint paths are available. Although the increases are slight as memory increases, the expected resilience does increase as more memory is added until the ideal situation occurs where all disjoint paths have only one intermediate node.
Figure 9 shows the communication overhead of each protocol as the network size increases. The resilience gains offered by MP-AP come at an increased communication cost since multiple paths must be used. Furthermore, the path found with P-AP will select intermediate nodes which have the fewest total physical hops, so the paths with MP-AP potentially find extremely poor intermediate nodes which are many physical hops away from the other nodes in the path. This difference in path selection enables MP-AP to be more resilient as an attacker cannot actively modify routing information to become part of the path.

Figure 10 shows the communication overhead as a function of density. The density has two impacts in terms of communication. Firstly, as these protocols aim to establish D-AP has a small amount of overhead locally, but we ignore this overhead as it is negligible compared to performing multi-hop communication in the path and multi-path based protocols.
Connected graph: Proportion insecure links

Figure 11 shows all pairwise and connected graph versions for each protocol given varied proportions of compromised nodes. We first observe that between D-CG and D-AP, the D-CG can increase resilience, but it still converges to all links being compromised quite quickly. We can see the relationship between these two protocols better when we vary density in our next scenarios. The path and multipath based protocols do not offer noticeable increases in resilience in their connected graph versions versus the all pairwise versions. These protocols get their advantages in terms of communication overhead.

Figure 12 shows all pairwise and connected graph versions given varied density. The varied density has significant impact on the connected graph protocols as it directly affects how many links must be established, that is, the higher density the fewer potential links must be established since each node has more chances to have a physical neighbor that it can share a key with. Again the path and multi-path based protocols exhibit little difference in resilience between the connected graph and all pairwise establishment.
Connected graph: Communication overhead

![Communication overhead graph](image)

**Key points.**

- For the direct protocols, D-AP and D-CG, the higher the physical density the greater resilience in the D-CG protocol compared with D-AP.
- For the path and multi-path protocols, P-AP, P-CG, MP-AP, and MP-CG, the P-CG and MP-CG gain advantages when physical density is high when compared with P-AP and MP-AP protocols.

### 7.4. Varied number of paths in MP-AP

We have shown our protocol compared with relevant existing work, and we now investigate the behavior of our protocol when altering the number of paths being used for key establishment. We focus on odd values as these are ideal for resilience which can be seen if you compare MP-AP-3 with MP-AP-4 since they both are broken when two paths are broken while the latter has a higher cost of four paths.

Figure 15 reveals that using more paths offers increased resilience even as the number of compromised nodes increases. At the most extreme scenario, MP-AP-9 can ensure that half of the links become insecure when compared with MP-AP-3. The exact...
Multi-path

Figure 16 shows resilience when varying the number of paths for each scheme. Up until 1250 nodes, the trend is using more paths results in higher resilience. However, this trend is broken in larger networks. The issue that arises with the protocols with more paths in larger networks is that the logical topology being constructed will offer fewer and fewer disjoint paths of small hop-length. Thus, when enough disjoint paths are sufficiently long the resilience drops significantly since just one intermediate logical node must be malicious to corrupt the entire path.

Figure 17 shows the impact of memory on security of the protocols. The memory directly affects the density used in the random logical topologies. Increasing density will ensure more, shorter disjoint paths which is what affects resilience. We observe
Summary

- Network coding brings new challenges and opportunities

Challenge

- Defenses against particular types of attacks against network coding: pollution

Opportunity

- Design of key management for sensor networks that leverage network coding and multi-path